REMARKS

Claims are pending in the application, claims being canceled and claims being newly added herein. Claims are the only independent claims.

Drawings

The drawings stand objected to under 37 C.F.R. § 1.83(a) as failing to show every feature set forth in the claims. In particular, the Examiner requires that the drawings show the "modifying an object on said display in response to instructions specified by a respective active virtual thread in said second linked list" (independent claim 21) or the feature be canceled from the claim.

In response to the objection to the drawings under 37 C.F.R. § 1.83(a), applicant encloses herewith a photocopy of the first sheet of drawings, including Figures 1-3, showing a proposed change to Figure 2 in red ink. The proposed change schematically shows display 16 with a solid-line rectangle labeled with reference numeral 202. Behind the solid-line rectangle is shown a phantom-line rectangle labeled with reference numeral 204.

Page 29 of the specification has been amended to insert the reference numerals 202 and 204. The rectangle 202 in Figure 2 represents a button in one configuration, while the rectangle 204 represents the same button in a modified configuration.

In anticipation of the Examiner's approval of the drawing change, applicant encloses herewith a photocopy of the first sheet of drawings, including Figures 1-3, showing the proposed change to Figure 2 in black ink.

Claim Objections

Claim 44 stands objected to because of the word "byte-".

In response to this objection, applicant has amended claim 44 to change the hyphenated half-word "byte-" to the word "bytecode".

Claims Rejections - 35 U.S.C. § 112

Claims 1-45 stand rejected under 35 U.S.C. § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

a. The Examiner specifically maintains that the following terms lack antecedent basis: (i) "the computer" in claim 41, lines 16-17, (ii) "local variables" in claim 44, lines 17-18, and (iii) "said virtual program threads" in claim 45, line 14.

Applicant respectfully traverses the rejection of claim 41 under 35 U.S.C. § 112, second paragraph, for lack of antecedent basis. The term "the computer" in lines 16 and 17 refers back to the recitation of the computer in the preamble of the claim. To change the recitation in lines 16 and 17 to "a computer" would be confusing in that one would not know that it is the same computer as recited in the preamble.

Applicant respectfully traverses the rejection of claim 44 under 35 U.S.C. § 112, second paragraph, for lack of antecedent basis. The term "local variables" in the last line of the claim is proper. The term "local variables" does not include the definite article "the" or the modifier "said" and thus does not imply that there is a prior recitation of "local variables" in the claim. This is proper, since there is no prior recitation of "local variables" in the claim. It is to be noted that first introduction of a term such as "local variables" in the plural does not and cannot have a modifier article that corresponds to the article "a" for the singular.

In response to the rejection of claim 45 under 35 U.S.C. § 112, second paragraph, for lack of antecedent basis, that claim has been amended to change the term "said virtual program threads" to the term "said virtual threads". The term as modified has proper antecedent basis.

b.iv. The Examiner maintains that the term "respective" in lines 10 and 11 of claim 1 is indefinite because it purportedly is not made explicitly clear in the claim language what this is respective to. According to the Examiner, it is also not made

explicitly clear in the claim language whether "a respective one" (line 11 is the same as the "respective virtual thread" of line 7 or if a separate respective one is created.

Applicant respectfully traverses the Examiner's rejection of claim 1 on the grounds that the term "respective" in lines 10 and 11 of claim 1 is indefinite. Applicant contends that the claim language is quite clear as to the meaning and antecedent intent.

The second step of claim 1 recites "for each of a plurality of tasks or jobs to be performed by the computer, automatically creating ***a respective virtual thread*** of execution context data including (a) a memory location of a next one of said pseudocode instructions to be executed in carrying out ***the respective task or job*** and (b) the values of any local variables required for carrying out the ***respective task or job***...." Each of the highlighted phrases refers back to the phrase "for each of a plurality of tasks or jobs" at the beginning of the step. Thus, each task or job has its own virtual thread. Each task or job has a corresponding memory location for the next pseudocode instruction to be carried out. Each task or job may have local variables peculiar to it that are used for carrying out that task or job. This is what the plain language of the claim says. Claim 1 also states "a plurality of said tasks or jobs each entailing execution of ***a respective one of said pseudocode instructions*** comprising a plurality of machine language instructions." This phrase plainly and straightforwardly states that at least some of the tasks or jobs ("a plurality of said tasks or jobs") are <u>each</u> accomplished by carrying out an associated pseudocode instruction that comprises a plurality of machine instructions. Each of these tasks or jobs is associated with or assigned a matching pseudocode instruction that comprises a plurality of machine instructions. This is what the plain language of the claim means.

With respect to the phrase "a respective one", the Examiner is confused because he is not reading the phrase in its proper context. The phrase in its entirety reads "a respective one of said pseudocode instructions." This is clearly not the same meaning that the phrase "a respective one of said virtual threads" would have. The entire clause,

which reads "a plurality of said tasks or jobs each entailing execution of a respective one of said pseudocode instructions comprising a plurality of machine language instructions" has the meaning indicated above.

b.v. The Examiner repeats the above-stated grounds of rejection for claims 21, 31, 41, and 44. In general, in response to this rejection, applicant points out that the use of the term "respective" is generally in a clause referring to each of a plurality of first items. Each item is then associated with a corresponding one of a plurality of second items. Thus, the first items are associated with respective ones of the second items.

Specifically, in response to the rejection of claim 21 under 35 U.S.C. § 112, second paragraph, owing to the word "respective" as used in that claim, applicant has amended claim 21 herein to clarify the referent and meaning of the word "respective". In particular, claim 21 has been amended to recite "each given one of said threads including context or state data, a mutex and a pointer to a next thread in the respective list in which said given one of said threads is listed...." Claim 21 has been further amended to recite "said interpreter being operatively connected to said display in part for modifying an object on said display in response to instructions specified by a respective active virtual thread in said second linked list associated with or assigned to said object." These amendments to claim 21 are believed to adequately clarify the use of the word "respective" in that claim.

With respect to claim 31, applicant respectfully traverses the rejection of that claim under 35 U.S.C. § 112, second paragraph, owing to the word "respective" as used in that claim. Applicant respectfully asserts that the word's meaning is sufficiently clear for one of ordinary skill in the art to understand the scope of the claim. In the clause "for each task of a plurality of tasks to be performed by the computer, using the interpreter to define a respective virtual thread" it is clear that the word "respective" refers back to the phrase "each task" and thus identifies a virtual thread associated with each task. Each taks has a thread, called "the respective virtual thread". Similarly, in the clause, "during

each time slice of a series of consecutive time slices, executing bytecode instructions of a respective current thread selected from among the virtual threads" the word "respective" associates the current thread with a time slice. Each time slice corresponds to its own current thread, its ***respective*** virtual thread.

In response to the rejection of claim 41 under 35 U.S.C. § 112, second paragraph, owing to the word "respective" as used in that claim, applicant has amended claim 41 herein to clarify the referent and meaning of the word "respective". Claim 21 now reads, in relevant part: "to execute, during each time slice of a series of consecutive time slices, bytecode instructions of a respective current thread selected from among the virtual threads". The word "respective" as used in this claim associates each time slice with a respective current thread. The claim has been amended to insert the antecedent basis "each time slide ..." ahead of the phrase "respective current thread" to clarify the antecedent basis of the term "respective current thread". Each time slice sees the execution of a current thread peculiar to that time slice, i.e., its ***respective*** current thread.

With respect to claim 44, applicant respectfully traverses the rejection of that claim under 35 U.S.C. § 112, second paragraph, owing to the word "respective" as used in that claim. Applicant respectfully asserts that the word's meaning is sufficiently clear for one of ordinary skill in the art to understand the scope of the claim. In the clause "operating an interpreter of said computer to assign computing tasks to respective virtual threads," it is clear that each of a plurality of computing tasks has a virtual thread assigned to it, that thread for any given computing task being the "respective virtual thread."

b.vi. The Examiner rejects claim 21 under 35 U.S.C. § 112, second paragraph, on the grounds that the word "object" is indefinite because it is not made explicitly clear in the claim language whether this refers to an item or a data software object.

Applicant respectfully traverses this rejection of claim 21. The claim explicitly states that the object is on a computer display. Thus, anything that appears in some form

on a computer display is an object within the meaning of claim 21, regardless of whether it is graphical such as an image of a geometrical object or whether it is data or alphanumeric information. In any event, the "object" is something that must naturally have a referent or control in the computer program, since the computer program controls what is shown on the display. To the extent that an "item" does not appear on the computer display, then it is not an object within the meaning of claim 21. If an item or a data software object appears in some form on the computer display, then it is an object according to claim 21. If the item or data software object does not appear in any form on the computer display, then it is not an object within the meaning of the claim.

b.vii. The Examiner rejects claim 44 under 35 U.S.C. § 112, second paragraph,on the grounds that the word "interval" is purportedly indefinite because it is not made explicitly clear in the claim language how this distinguishes from a "time slice", if there is a difference between those terms.

In response to this rejection of claim 44 under 35 U.S.C. § 112, second paragraph, claim 44 has been amended herein to clarify that the described time check is against the predetermined duration of the time slice. If the time slice has run its course upon completed execution of a bytecode or pseudocode instruction, then a context switch is made and the processing of a next active virtual thread in a corresponding succeeding time slice is commenced. Conversely, if upon completed execution of a pseudocode instruction, the current time slice is not yet over, then another pseudocode instruction is processed with respect to the same virtual thread.

b.viii. The Examiner rejects claim 45 under 35 U.S.C. § 112, second paragraph, on the grounds that the word "objects" is indefinite because it is not made explicitly clear in the claim language whether this refers to an item or a data software object.

Applicant respectfully traverses this rejection of claim 45 for the reasons discussed above with respect to claim 21.

b.ix. The Examiner rejects claim 21 under 35 U.S.C. § 112, second paragraph, on the grounds that there is no structural relationship established between the memory and the interpreter. Specifically, the Examiner maintains that it is not made clear what the bytecode or pseudocode instructions have to do with the state and context data of the virtual threads because a structural relationship has not been established between them.

Applicant respectfully traverses the Examiner's position that claim 21 fails to set forth a "structural relationship established between the memory and the interpreter." Claim 21 explicitly refers to "said interpreter being operatively connected to said memory." That recitation delineates a structural relationship.

In further response to this rejection under 35 U.S.C. § 112, second paragraph, claim 21 has been amended to recite that the interpreter is operatively connected to the memory "for executing said bytecode or pseudocode instructions according to context and state data of different virtual threads in said second linked list ...." Claim 21 as so amended sets forth a structural relationship between the bytecode or pseudocode instructions, on the one hand, and the context and state data of the virtual threads, on the other hand.

b.x. The Examiner rejects claim 41 under 35 U.S.C. § 112, second paragraph, on the grounds that there is no structural relationship established between the memory and the interpreter.

In response to the rejection of claim 41 under 35 U.S.C. § 112, second paragraph, that claim has been amended herein to explicitly set forth a structural relationship between the memory and the interpreter. Specifically, claim 41 now recites that the interpreter is operatively connected to the memory.

b.xi. The Examiner rejects claim 44 under 35 U.S.C. § 112, second paragraph, on the grounds that there is no structural relationship established between the memory and the interpreter. Specifically, the Examiner maintains that it is not made clear what the

bytecode or pseudocode instructions have to do with the state and context data of the virtual threads because a structural relationship has not been established between them.

Applicant respectfully traverses the rejection of claim 44 under 35 U.S.C. § 112, second paragraph. Claim 44 does not recite a memory. Accordingly, it would not make sense to define a relationship in that claim between an interpreter and a memory. Moreover, claim 44 is a method claim. It is not necessary to define structural relationships in a method claim.

With respect to the relationship between the bytecode or pseudocode instructions, on the one hand, and the state and context data of the virtual threads, on the other hand, it is believed that claim 44 sets forth sufficient relationship the meet the requirements of Section 112, second paragraph, of the Patent Statute. Claim 44 recites "in each of said time slices, additionally operating said interpreter to execute selected ones of said bytecode or pseudocode instructions pursuant to the state and context data of a current one of said virtual threads." The state and context data specify information that enables he interpreter to operate on the pseudocode instructions.

Claims Rejections - 35 U.S.C. §§ 102 and 103

Claims 1-18, 20, and 31-43 stand rejected under 35 U.S.C. § 103(a) 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 6,766,515 to Bitar et al. in view of U.S. Patent No. 6,260,150 to Diepstraten et al.

Claim 19 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over Bitar et al. in view of Diepstraten et al. and further in view of U.S. Patent No. 4,744,048 to Blanset et al.

Claims 44 and 45 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Bitar et al. in view of Blanset et al.

The Examiner has indicated that claims 21-30 would be allowable if rewritten to overcome the rejection under 35 U.S.C. § 112, second paragraph.

Claim 1 Applicant respectfully traverses the rejection of claim 1 under35 U.S.C. § 103(a). Claim 1 recites a method for operating a computer comprising storing in a computer memory a plurality of ***pseudocode instructions***, at least some of the pseudocode instructions comprising a plurality of machine code instructions. Applicant respectfully traverses the rejection of claim 1 under35 U.S.C. § 103(a). Claim 1 recites a method for operating a computer comprising storing in a computer memory a plurality of ***pseudocode instructions***, at least some of the pseudocode instructions comprising a plurality of machine code instructions. For each of a plurality of tasks or jobs to be performed by the computer, a respective virtual thread of execution context data is automatically created which includes (a) a memory location of a next one of the ***pseudocode instructions*** to be executed in carrying out the respective task or job and (b) the values of any local variables required for carrying out the respective task or job, a plurality of the tasks or jobs each entailing execution of a respective one of the ***pseudocode instructions*** comprising a plurality of machine language instructions. The method of claim 1 additionally comprises processing each of the tasks or jobs in a respective series of time slices or processing slots under the control of the respective virtual thread, and in every context switch between different virtual threads, undertaking such context switch only after completed execution of a currently executing one of the pseudocode instructions.

The Bitar patent never once mentions pseudocode instructions. The patent specifically describes a mechanism based on saving the registers and other processor execution state for each thread and switching between them. The Bitar reference more specifically describes time slicing via conventional threading using conventional processor instructions. There is no suggestion in this reference of virtual threading, which uses pseudocode instructions.

This is an important differentiation because pseudocode instructions are interpreted by an interpreter program (which maps them in *real-time* to machine language

instructions). Pseudocode instructions are not interpreted directly by the processor. This means that virtual threading is not concerned with time slicing or the state of the processor's registers.

Claim 1 is not obvious from the reference relied on by the Examiner in rejecting that claim under Section 103 of the Patent Statute. The secondary reference relied on by the Examiner, Diepstraten et al., also fails to mention bytecode or pseudocode or interpreters and does not concern virtual threading.

According to the Examiner (paragraph 7 of the Office Action), Bitar fails to explicitly teach performing context switching only after completed execution of a currently executing one of said pseudocode instructions. The Examiner maintains, however, that Diepstarten teaches time slice context switching to occur at the end of every instruction (col. 8, lines 42-49 and 63-66). The Examiner then holds that it would have been obvious to one of ordinary skill in the art at the time the invention was made to include the feature of time slice context switching to occur at the end of every instruction to the existing system of Bitar because it is preferred to switch states or instructions when the instruction is over/completed (col. 8, lines 50-/66).

Applicant respectfully disagrees with the Examiner's reading of Diepstraten. Diepstarten does teach context switching after every instruction. However, it is irrelevant because Diepstarten is talking about machine language instructions ***not*** pseudocode instructions.

If one were (as Diepstarten suggests) to perform time slicing after every machine language instruction, the application would become so slow as to be unusable for any practical application.

What claim 1 refers to is performing a time slice after every ***pseudocode*** instruction. A pseudocode instruction is a significantly larger in scope than a machine language instruction. Time slice context switching operates at the machine language instruction level. This causes a problem because a set of machine language instructions could be interrupted at any point for a context switch, which means that data synchronization must be performed even for very simple tasks.

With virtual threading, context switching is done using a pseudocode instruction count rather than a time slice. This guarantees that a pseudocode instruction executes atomically (without interruption). This means that within a pseudocode instruction, no data synchronization is necessary (which makes a much cleaner programming technique possible, as well as resulting in faster execution time). The machine language instructions that comprise a pseudocode instruction are generally of a large enough set to prevent performance degradation from too-frequent context switches, but small enough to make the context switching fine-grained enough to successfully emulate simultaneous processes.

Claim 3 Applicant respectfully traverses the rejection of claim 3 for the reasons set forth above with respect to claim 1 and for additional reasons discussed here. Claim 3 depends from claim 2 which in turn depends from claim 1. Claim 2 states that each of the virtual threads is part of a respective linked list of virtual threads, each of the virtual threads further including a pointer to a next virtual thread in the respective linked list, the method of claim 1 further comprising, for every context switch between different virtual threads, consulting the pointer of a currently executing virtual thread to determine an identity of a next virtual thread to be executed. As set forth in claim 3, the respective

linked list is one of a plurality of linked lists of the virtual threads, one of the linked lists being a list of idle virtual threads, another of the linked lists being a list of active virtual threads, an additional one of the linked lists being a list of queued virtual threads, the method further comprising periodically moving at least one virtual thread from the list of queued virtual threads to the list of active virtual threads.

According to the Examiner, Bitar teaches that the respective linked list is one of a plurality of linked lists of the virtual threads, one of the linked lists being a list of idle virtual threads, another of the linked lists being a list of active virtual threads, an additional one of the linked lists being a list of queued virtual threads, and further teaches periodically moving at least one virtual thread from the list of queued virtual threads to the list of active virtual threads (cal. 11, lines 1-7, col. 12, lines 7-21).

Applicant respectfully contravenes the Examiner's assertions about the techings of Bitar. Bitar does not discuss this. Instead, he discusses assigning ***physical*** (not virtual) user-space threads to different kernel-space threads. Given hindsight knowledge of other (non-obvious) aspects of the present application, assigning virtual threads using a similar technique may become obvious as a way of implementing the present invention, but that would be predicated upon an understanding on the non-obvious portions of the patent.

Claim 8 Claim 8 recites that each of the virtual threads of claim 1 is assigned a message queue and that the method of claim 1 further comprises entering a message in a message queue of a selected one of the virtual threads during execution of a task or job pursuant to another one of the virtual threads.

According to the Examiner, Bitar teaches the subject matter of claim 8 at col. 17,

lines 22-35 and col. 18, lines 21-41.

Applicant respectfully asserts that the Examiner is mistaken as to the teachings of Bitar in this regard. Bitar does ***not*** refer to assigning message queues to threads. In fact, the only reference to messages in the entire Bitar patent is a brief reference to the need to use messages to communicate between multiple kernels (i.e., on a multi-processor system). Even in that case, the Bitar reference does not describe what the messages are or how they are used. In any case, as far as the Bitar reference is concerned, messages are not a means of inter-thread communications. This is completely different from applicant's approach, where messages are used as a means of inter-thread communication, so that different virtual threads can exchange data with each other without the need to worry about specific data synchronization issues. In fact, ***Bitar never talks about a message queue***. The only queue described by Bitar is a queue of scheduled threads.

Claim 10 Claim 10 depends from claim 8 and further recties that the selected one of the virtual threads and the another one of the virtual threads are proxy or interface threads on different computers, the entering of the message in the message queue including transmitting the message over a communications link between the computers.

According to the Examiner, Bitar teaches this subject matter at col. 8, lines 28-3 1.

Applicant respectfully but strenuously traverses this position of the Examiner. Bitar ***never*** refers to proxy or interface threads for other computers. There is a ***single*** reference to "network" in the entire Bitar patent, and that refers to applying the invention of that prior art patent to a distributed computer system using ***shared memory*** across the

network (which is impractical over the internet due to the high latency inherent in the internet). Applicant's approach specifically addresses that shortcoming by using proxy threads.

Claim 11 Claim 11 modifies the subject matter of claim 1 and recites that the creating of the virtual threads, the processing of the tasks or jobs in respective series of time slices or processing slots, and the undertaking of context switches all include the operating of the computer under an interpreter program.

In rejecting claim 11, the Examiner maintains that Bitar teaches using an interpreter program (col. 1, lines 26-35).

This is not accurate. Bitar never once mentions an interpreter program. The ***only*** use of the word "virtual" in that prior art reference is in a completely different context and a completely different meaning than applicant's use of the word. The Bitar patent talks about virtual resources such as: file handles, processors, memory, etc. Each of these virtual resources is ***directly mapped*** to a physical resource (a physical processor, for example) and no interpretation takes place. It is simply a 1:1 lookup table (for example, Virtual File #1 = Physical File #6... or virtual processor #2 = Physical processor #1). This is done for one reason only: In case the running thread is moved to a different processor, the identifiers it uses to refer to these resources remain valid because the mappings are changed. This bears absolutely no relation to applicant's approach as set forth in claim 11.

Claim 12 Claim 12 depends from claim 11 and recites that the method of that claim further comprises running a plurality of instances of the interpreter program on the computer, each instance corresponding to a native thread, where each native thread

creates a respective set of virtual threads of execution context data, processes each of a plurality of tasks or jobs in a respective series of time slices or processing slots under the control of the respective virtual thread, and in every context switch between different virtual threads, undertakes such context switch only after completed execution of a currently executing one of the pseudocode instructions.

The Examiner rejects claim 12 for the same reasons as stated in the rejection of claim 1. In addition, the Examiner contends that Bitar teaches a native thread that a virtual thread is mapped from (col. 13, lines 30-49).

Applicant respectfully traverses the Examiner's contention with respect to the teachings of Bitar in relation to the specific subject matter of claim 12. The Bitar reference never mentions virtual threads. That reference talks about user-space threads, but these threads are still conventional threads; they simply exist in user-space, and map many-to-one to kernel-space threads. As far as the instant application is concerned, both kernel-space and user-space threads are ***physical*** threads in that the instruction pointer in the thread refers to the next ***machine-language*** instruction to execute and the thread context includes processor register data, and time slicing is done at the machine language instruction level. Pursuant to the present invention, the instruction pointer refers to the next **pseudocode instruction**, which is mapped ***in real time*** by an interpreter to a plurality of machine language instructions. This provides major benefits (mentioned above). The closest Bitar comes to pseudocode instructions is a passing reference to compiler-driven scheduling. However, this is a passing reference and does not actually address ***how*** it could be done.

Claim 20 Claim 20 depends directly from claim 1 and further recites that the

time slots or processing slots are measured by counting consecutively executed pseudocode instructions. In addition, claim 20 states that the method of claim 1 further comprises, for each of a plurality of the time slices or processing slots, terminating the respective time slot or processing slot upon counting a predetermined number of consecutively executed pseudocode instructions.

According to the Examiner, the Bitar reference teaches that time slots or processing slots are measured by counting consecutively executed pseudocode instructions and also teaches, for each of a plurality of said time slices or processing slots, terminating the respective time slot or processing slot upon counting a predetermined number of consecutively executed pseudocode instructions (time quantum expires) (col. 2, lines 22-34).

Applicant respectfully traverses the Examiner's characterization of the Bitar reference. That reference refers to solely to ***machine language*** instructions, not ***pseudocode instructions***. This actually applies to ***all*** of the examiner's references to Bitar. There is no reference whatsoever in that patent to pseudocode instructions.

Claim 19 Claim 19 depends directly from claim 1 and sets forth that the tasks or jobs processed in respective series of time slices or processing slots under the control of the respective virtual threads include (a) controlling objects imaged on a computer display, each of the objects constituting a separate task or job assigned a respective one of the virtual threads, and (b) monitoring actuation of keys on a computer keyboard, each of the keys constituting a separate task or job assigned a respective one of the virtual threads.

In his rejection of claim 19, the Examiner admits that Bitar in view of Diepstraten fails to explicitly teach controlling objects imaged on a computer display and monitoring actuation of keys on a computer keyboard. The Examiner contends, however, that Blanset teaches controlling objects imaged on a computer display and monitoring the input of keys on a keyboard (col. 13, lines 45-60 and see Abstract) and that it would have been obvious to one of ordinary skill in the art at the time the invention was made to include the feature of controlling objects imaged on a computer display and monitoring the input of keys on a keyboard to the existing system of Bitar and Diepstraten because it would allow for context switching by the user (col. 1, lines 6-26).

Applicant respectfully traverses the rejection of claim 19 under 35 U.S.C. § 103(a) as being unpatentable over Bitar et al. in view of Diepstraten et al. and further in view of Blanset et al.

Obviously, controlling objects imaged on the display is useful (in fact, ever since computers have had displays, nearly ***all*** programs have done exactly this. However, that is not what applicant discloses in his application. Applicant's virtual threading approach (and virtually unlimited number of threads) makes it possible to assign a separate virtual thread to ***each and every on-screen object***. This has never been done before, and none of the patents relied on by the Examiner even touches on it. This allows for instant response to the user when an object is clicked on (for example) because there is a thread dedicated to that object, always ready to instantly handle the input.

Applicant has considerable data showing the importance of instant graphic user interface response. Slow response degrades information retention and increases human error rates and frustration with the software.

Typically, for a program to ensure instant response to the user, the developer of that program must go to great lengths to structure the program in a fashion that allows the program to check for user input on an extremely frequent basis. This results in a much more complex program that is significantly more difficult to develop and debug.

Because virtual threading makes it possible to assign a separate thread to every single user interface object realized on the screen, it is possible to have instant response without any increase in code complexity. In fact, it reduces code complexity by eliminating the need for extra application code to retain the state of the user interface object (the state is automatically preserved as part of the thread context in the virtual threading mechanism because each object has its own thread).

Claim 31 Independent claim 31 claims a multitasking method in a computer having an interpreter for executing a series of bytecode instructions each consisting of a multiplicity of machine code steps. The multitasking method comprises using the interpreter for each task of a plurality of tasks to be performed by the computer to define a respective virtual thread, executing, during each time slice of a series of consecutive time slices, bytecode instructions of a respective current thread selected from among the virtual threads, and executing a context switch from one of the virtual threads to another of the virtual threads only after execution of one of the bytecode instructions.

Applicant respectfully traverses the Examiner's rejection of claim 31 under 35 U.S.C. § 103(a). Claim 31 recites a method for operating a computer comprising operating an ***interpreter*** in a computer to execute, during each time slice of a series of consecutive time slices, ***bytecode instructions*** of a respective current thread selected from among the virtual threads, and executing a context switch from one of the virtual threads

to another of the virtual threads only after execution of one of the bytecode instructions. Neither the Bitar reference nor the Diepstraten reference mentions an interpreter or bytecode instructions, let alone executing a context switch from one virtual thread to another only after execution of one of the bytecode instructions.

Claim 44 According to independent claim 41, a multi-tasking computer comprises a memory storing state and context data of multiple threads or tasks and an interpreter operatively connected to the memory for executing a series of bytecode instructions each consisting of a multiplicity of machine code steps. The interpreter is programmed to define a respective virtual thread for each task to be performed by the computer, to execute, during each time slice of a series of consecutive time slices, bytecode instructions of a respective current thread selected from among the virtual threads, and to execute a context switch from one of the virtual threads to another of the virtual threads only after execution of one of the bytecode instructions.

Claim 44 distinguishes over the art relied on by the Examiner for the same reasons discussed above with reference to claim 31.

Claim 44 As set forth in claim 44, a computer method comprises running a timer of a computer to generate a series of time slices or processing slots, compiling input user source code into bytecode or pseudocode instructions each corresponding to a multiplicity of machine code instructions, and operating an interpreter of the computer to assign computing tasks to respective virtual threads, the assigning of the computing tasks to the virtual threads including identifying and storing state and context data for each of the computing tasks. The method further comprises in each of the time slices, additionally operating the interpreter to execute selected ones of the bytecode or

pseudocode instructions pursuant to the state and context data of a current one of the virtual threads. During a current one of the time slices, after the execution of each successive one of the selected bytecode or pseudocode instructions and only after such execution, the interpreter is further operated to check whether the current one of the time slices has elapsed or terminated since a commencement of execution of instructions pursuant to the current one of the virtual threads. Upon a determination of elapsing of the current one of the time slices, the interpreter is operated to perform a context switch.

According to the Examiner, the Bitar reference teaches a computer method comprising compiling input user source code into byte- or pseudocode instructions each corresponding to a multiplicity of machine code instructions (col. 5, lines 32-57), and operating an interpreter of the computer to assign computing tasks to respective (mapped) virtual threads, the assigning of the computing tasks to the virtual threads including identifying and storing state and context data for each of the computing tasks (col. 1, lines 25- 35, col. 4, lines 57-67, col. 5, lines 19-25, and col. 6, lines 9-12).

Applicant respectfully traverses the Examiner's interpretation of the Bitar reference with respect to the subject matter of claim 44. Other than a single passing reference to compiler-driven scheduling, the Bitar reference ***never mentions compilers or interpreters.***

Conclusion

For the foregoing reasons, independent claims 1, 31, 41, and 44, as well as the claims dependent therefrom, is deemed to be in condition for allowance. The Examiner has additionally indicated that claim 21 distinguishes over the art. The rejection of that claim under 35 U.S.C. § 112, second paragraph, is believed to have been overcome by the

arguments and amendments made herein. An early Notice passing the application is earnestly solicited.

Should the Examiner believe that direct contact with applicant's attorney would advance the prosecution of this application, the Examiner is invited to telephone the undersigned at the number below.

Respectfully submitted,

COLEMAN SUDOL SAPONE, P.C.

By: _____

R. Neil Sudol
Reg. No. 31,669

714 Colorado Avenue
Bridgeport, CT 066-05-1601
(203) 366-3560

Dated: February 28, 2005

FIG. 1

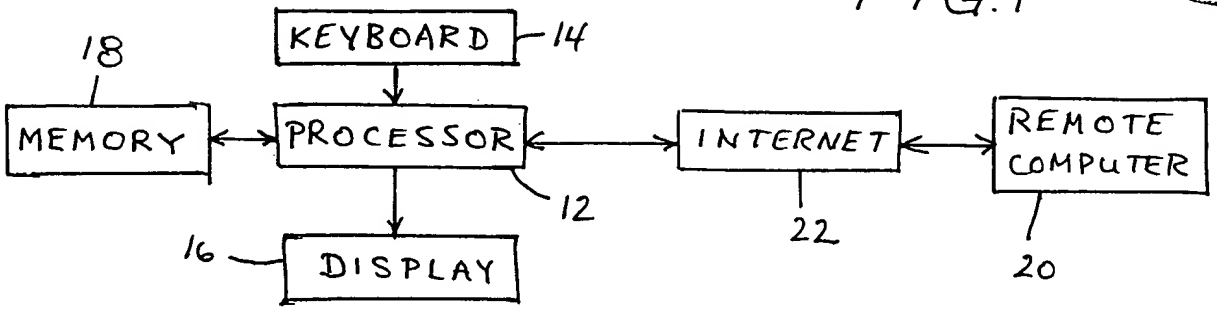

FIG. 2

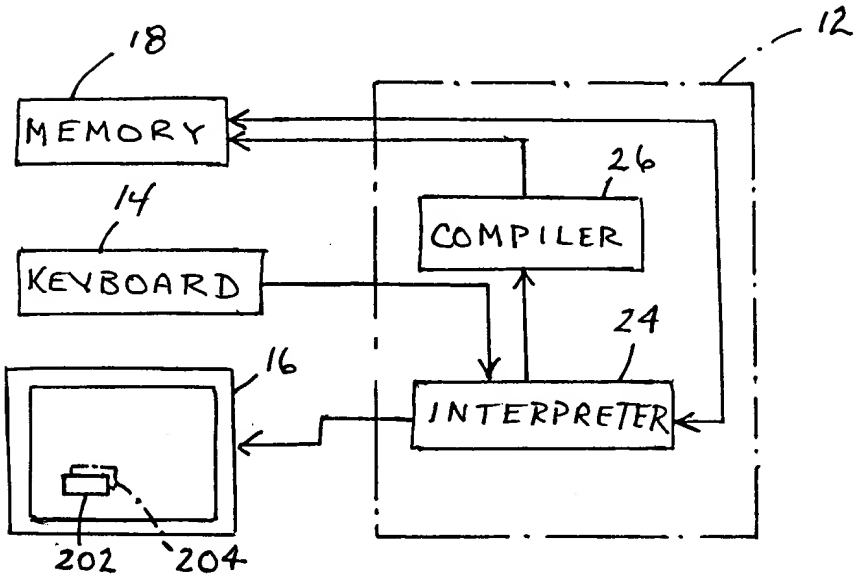

FIG. 3